UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/756,579	01/08/2001	John L. Reid	INTL-0463-US (P9817)	5624

7590 10/31/2005

Timothy N. Trop
TROP, PRUNER & HU, P.C.
STE 100
8554 KATY FWY
HOUSTON, TX 77024-1805

EXAMINER	
BULLOCK JR, LEWIS ALEXANDER	
ART UNIT	**PAPER NUMBER**
2195	

DATE MAILED: 10/31/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/756,579	REID, JOHN L.	
	Examiner	Art Unit	
	Lewis A. Bullock, Jr.	2195	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *29 July 2005*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-20* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-20* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *08 January 2001* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

Claim Rejections - 35 USC § 103

1. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims 1-20 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Application Isolation in the Java Virtual Machine" by CZAJKOWSKI.

As to claim 1, CZAJKOWSKI teaches a method comprising: running at least two applications (applications); enabling the applications to share a class (class having static variables); duplicating member data (static fields of the class) for the class for each application (application) in a shared memory (table shared between the applications that contains the copies of Counter$sFields); and providing an identifier (application id) to each application (application) to enable each application to access its member data (copy of static fields) in the shared memory (shared table) (pg. 356, "Consider a class X, containing static fields...X$aMethods maintains an instance of X$sFields for each application; the methods of X$aMethods access the correct instance of X$sFields based on the application id extracted from the current thread...The second generated class is Counter$aMethods. It contains a table mapping application identifiers onto per-application copies of Counter$sFields...Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated yet and that the appropriate initialization has

to be taken care of."; pg 363, "The runtime modifications operate as follows. At load time, each class is examined and each static field is replicated n times (n is the maximum allowed number of applications...Fetching an application id and using it to index an array of copies of a static field translates into less than ten machine instructions...When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment. Whenever a class is used by this application for the first time, the static initializer is run (when present), initializing the correct replicas of static fields."). However, CZAJKOWSKI does not explicitly teach that the identifier is a handle. It is well known in the art that a handle is any token that a program can use to identify and access an object such as a device, a file, a window, or a dialog box. CZAJKOWSKI teaches the application id is used to access the correct static fields as disclosed above. Therefore it would be obvious to one of ordinary skill in the art at the time of the invention that the identifier is a handle.

As to claim 2, CZAJKOWSKI teaches enabling each application to use a shared memory (via a table shared by the applications to access the correct Counter$sField copy) (pg. 356, "Consider a class X, containing static fields...X$aMethods maintains an instance of X$sFields for each application; the methods of X$aMethods access the correct instance of X$sFields based on the application id extracted from the current

thread...The second generated class is Counter$aMethods. It contains a table mapping application identifiers onto per-application copies of Counter$sFields...Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated ye and that the appropriate initialization has to be taken care of."; pg 363, "The runtime modifications operate as follows. At load time, each class is examined and each static field is replicated n times (n is the maximum allowed number of applications...Fetching an application id and using it to index an array of copies of a static field translates into less than ten machine instructions...When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment. Whenever a class is used by this application for the first time, the static initializer is run (when present), initializing the correct replicas of static fields.").

As to claim 3, CZAJKOWSKI teaches enabling each application (application) to define an address space of the shared memory (part of the table that stores the applications static field data) specific to each application (via the initialization of the static field copy for each application such that the correct copy is determined based on the applications identifier) (pg. 356, "Consider a class X, containing static fields...X$aMethods maintains an instance of X$sFields for each application; the methods of X$aMethods access the correct instance of X$sFields based on the application id extracted from the current thread...The second generated class is

Counter$aMethods. It contains a table mapping application identifiers onto per-application copies of Counter$sFields...Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated ye and that the appropriate initialization has to be taken care of."; pg 363, "The runtime modifications operate as follows. At load time, each class is examined and each static field is replicated n times (n is the maximum allowed number of applications...Fetching an application id and using it to index an array of copies of a static field translates into less than ten machine instructions...When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment. Whenever a class is used by this application for the first time, the static initializer is run (when present), initializing the correct replicas of static fields.").

As to claim 4, CZAJKOWSKI teaches duplicating (copy) process specific data (static fields) for each application (application) (pg. 356, "Consider a class X, containing static fields...X$aMethods maintains an instance of X$sFields for each application; the methods of X$aMethods access the correct instance of X$sFields based on the application id extracted from the current thread...The second generated class is Counter$aMethods. It contains a table mapping application identifiers onto per-application copies of Counter$sFields...Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named

field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated ye and that the appropriate initialization has to be taken care of."; pg 363, "The runtime modifications operate as follows. At load time, each class is examined and each static field is replicated n times (n is the maximum allowed number of applications... Fetching an application id and using it to index an array of copies of a static field translates into less than ten machine instructions... When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment. Whenever a class is used by this application for the first time, the static initializer is run (when present), initializing the correct replicas of static fields.").

As to claim 5, CZAJKOWSKI teaches automatically (during run-time) duplicating (copy) process specific data (static fields) in address space specific to each application ((pg. 356, "Consider a class X, containing static fields... X$aMethods maintains an instance of X$sFields for each application; the methods of X$aMethods access the correct instance of X$sFields based on the application id extracted from the current thread... The second generated class is Counter$aMethods. It contains a table mapping application identifiers onto per-application copies of Counter$sFields... Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated ye and that the appropriate initialization has to be taken care of."; pg 363, "The runtime modifications

operate as follows. At load time, each class is examined and each static field is replicated n times (n is the maximum allowed number of applications...Fetching an application id and using it to index an array of copies of a static field translates into less than ten machine instructions...When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment. Whenever a class is used by this application for the first time, the static initializer is run (when present), initializing the correct replicas of static fields.").

As to claim 6, CZAJKOWSKI teaches defining a shared class (Counter$aMethods) and using the share class to execute an instance of a class (Counter$sFields) to share (pg. 356, "Consider a class X, containing static fields...X$aMethods maintains an instance of X$sFields for each application; the methods of X$aMethods access the correct instance of X$sFields based on the application id extracted from the current thread...The second generated class is Counter$aMethods. It contains a table mapping application identifiers onto per-application copies of Counter$sFields...Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated ye and that the appropriate initialization has to be taken care of."; pg 363, "The runtime modifications operate as follows. At load time, each class is examined and each static field is replicated n times (n is the maximum allowed number of applications...Fetching an application id and using it to index an

array of copies of a static field translates into less than ten machine instructions...When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment. Whenever a class is used by this application for the first time, the static initializer is run (when present), initializing the correct replicas of static fields.").

As to claim 7, CZAJKOWSKI teaches invoking a shareable interface (initializer function / Counter$aMethods functions) to obtain a handle (application id) (pg. 356, "The original class Counter, undergoes the following modifications. All static fields are removed from Counter. A new method, hidden$initializer(), is added. It contains a modified code of the static initializer of Counter. It is invoked whenever a new application uses static fields of Counter for the first time."; pg. 356, "In our particular case there is only one static field and thus Counter$aMethods has two such access methods: put$cnt() and get$cnt(). Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated yet and that the appropriate initialization has to be taken care of."; see also Figure 2, program code wherein int id – Thread.currentAppId(), wherein the identifier is retrieved from the application thread).

As to claim 8, CZAJKOWSKI teaches specifying the handle (application id) on each method call to resolve the context (static fields) of the handle (application id)

(wherein the application id is used to retrieve the correct static fields for the shared class) (pg. 356, "Consider a class X, containing static fields...X$aMethods maintains an instance of X$sFields for each application; the methods of X$aMethods access the correct instance of X$sFields based on the application id extracted from the current thread...The second generated class is Counter$aMethods. It contains a table mapping application identifiers onto per-application copies of Counter$sFields...Each of them looks up the copy of Counter$sFields corresponding to the current application and then accesses the named field. If the lookup does not succeed it means that this application's copy of Counter$sFields has not been generated ye and that the appropriate initialization has to be taken care of."; pg 363, "The runtime modifications operate as follows. At load time, each class is examined and each static field is replicated n times (n is the maximum allowed number of applications...Fetching an application id and using it to index an array of copies of a static field translates into less than ten machine instructions...When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment. Whenever a class is used by this application for the first time, the static initializer is run (when present), initializing the correct replicas of static fields.").

As to claims 9-16, reference is made to an article comprising a medium that corresponds to the method of claims 1-8 and is therefore met by the rejection of claims 1-8 above. Claim 9 further details a processor-based system. CZAJKOWSKI teaches that the teachings are used on a PALM device that has a power of 2.7 MIPS at

16.6MHz processor clock and a heap (pg. 362). Therefore, it would be obvious that the teachings are executed on a processor-based system (PALM device).

As to claims 17-20, reference is made to a system that corresponds to he method of claims 1-4 and is therefore met by the rejection of claims 1-4 above. Claim 17 further details the system having a processor and storage. CZAJKOWSKI teaches that the teachings are used on a PALM device that has a power of 2.7 MIPS at 16.6MHz processor clock and a heap (pg. 362). Therefore, it would be obvious that the teachings are executed on a processor-based system (PALM device).

Response to Arguments

3. Applicant's arguments filed July 29, 2005 have been fully considered but they are not persuasive. Applicant argued that the reference does not teach "duplicating the member data for the class for each application in a shared memory" because the reference does not discuss where the asserted member data is stored. The examiner disagrees. Czajkowski states, "In particular, in our basic design the heap is physically shared by applications (although logically it contains disjoint graphs of objects belonging to different application.)" (pg. 364, Runtime Modifications, 4th paragraph). In addition, Czajkowski teaches, "Fetching an application id and using it to index an array of copies of a static field translates into less than ten machine instructions." (pg. 363, 6. In Run-time Implementation, 1st paragraph). Therefore, the since the array is accessible by a

plurality of applications and stores the copies of static fields for the applications the array is shared memory, i.e. a shared heap that is physically shared by applications.

Applicant then states the reference does not teach providing a handle "to each application" to enable "each application" to access its member data because the cited reference uses a Counter$sMethods table that maps application identifiers to the per-application copies of Counter$sFields and the applications themselves are not able to access the member data because they are provided no handle. The examiner disagrees. First, the examiner would like to point out that Applicant seems to alluding that the handle is a direct reference to the member data. The claims do not make this assertion realize. The claims at best detail that a handle is provided to the application and is used in allowing the application to access member data. There is no indication that the handle is a direct reference to the member data and any interpretation of such would be against procedures disclosed in M.P.E.P. 2111 of reading limitations of the specifications into the claims. Czajkowski teaches providing an application identifier to each application and when a class is used by this application for the first time the static initializer is run, initializing the correct replicas of static fields (pg. 363, 6. In Run-time Implementation, 2nd paragraph, "When an application gets loaded into the modified KVM, it is assigned an application id or is rejected if no more application slots are available at the moment....initializing the correct replicas of static fields"). Therefore, Czajkowski teaches providing the applications with an identifier, therein a handle as detailed in the rejection such that the identifier is used to provide the application with access to its member data. Czajkowski also teaches, "Fetching an application id and

using it to index an array of copies of a static field translates into less than ten machine instructions." (pg. 363, 6. In Run-time Implementation, 1st paragraph), "Each access to a static field has to be replaced with the appropriate get$() or put$() method. (pg. 356, 3 Details, 8th paragraph), "...each application sees a separate copy of System.out, but internally this print stream is implemented as prefixing each new line with the id of its application." (pg. 358, 3.5 A typical Cases, 2nd paragraph). Therefore, the handle is used to enable access to member data of the application by the application since the application is accessing the shared class such that the identifier is used to correctly access and manipulate the correct static fields. The examiner has explained above that the member data is stored in shared memory. Therefore, the claim language is met as broadly disclosed in the claims and the rejection is maintained.

Relevant Cited Prior Art, but not Applied

U.S. Patent 6,694,346 and "Building a Java virtual machine for server applications: The Jvm on OS/390" by IBM Corporation are cited by the Examiner to further illustrate that the concept of a handle allowing direct access to a copy of shared data of a shared class, when or if claimed by Applicant would not be novel. The references details a virtual machine that executes multiple applications to reuse shared classes in a shared memory, memory 18, wherein the applications create a private heap within the shared memory 18 to store application specific program data and static variables related to the shared class such that each application has access to its own copy of the static variables of the shared class. It would be obvious that since the

applications created the private heaps in the shared memory that the applications have handles to there heaps.

Conclusion

4. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Lewis A. Bullock, Jr. whose telephone number is (571) 272-3759. The examiner can normally be reached on Monday-Friday, 8:30 - 5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng An can be reached on (571) 272-3756. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

LEWIS A. BULLOCK, JR.
PRIMARY EXAMINER

October 26, 2005